Despliegue de SVG

Ahora que ya estamos familiarizados con la estructura básica y los elementos de una imagen en formato SVG, cómo podemos comenzar a generar diagramas con nuestros datos?

Usted ha podido ver que todas las propiedades de los elementos se definen como *atributos*. Eso quiere decir que se incluyen como pares propiedad/valor dentro de cada etiqueta del elemento, de esta manera:

```
<element property="value"/>
```

Umm, esto se parece extrañamente a HTML!

```
<p class="eureka">
```

Ya hemos usado los métodos `append()` y `attr()` para crear nuevos elementos de HTML y asignarles sus atributos. Debido a que los elementos de SVG existen en el DOM, tal como los elementos de HTML, podemos usar `append()` y `attr()` de la misma forma para generar imágenes en SVG!

Cree el SVG

Primero, debemos crear el elemento SVG que va a contener todas nuestras figuras.

```
d3.select("body").append("svg");
```

Se acuerda cómo la mayoría de métodos en D3 devuelven una referencia al elemento del DOM sobre el cual actúan? Al crear una nueva variable `svg`, podemos capturar la referencia que se le pasa a `append()`. Esto se puede entender como que el `svg` no es una "variable" sino una "referencia que apunta al objeto SVG recién creado". Esta referencia nos va a ahorrar muchas líneas de código en el futuro. En vez de buscar ese SVG cada vez - como en `d3.select("svg")` – solo decimos `svg`.

```
svg.attr("width", 500)
.attr("height", 50);
```

De otra forma, se podría haber escrito todo en una línea de código:

```
var svg = d3.select("body")
        .append("svg")
        .attr("width", 500)
        .attr("height", 50);
```

Puede ver el ejemplo de este código. Al inspeccionar el DOM verá que ahí está, en verdad, el elemento SVG vacío.

Para simplificar, recomiendo que se incluyan los valores de ancho y altura como variables al comienzo del código, de esta manera (vea la fuente):

```
//Width and height
var w = 500;
var h = 50;
```

Esto se hará en todos los siguientes ejemplos. Al convertir en variables los valores de tamaño, es muy fácil hacer referencia a ellos a lo largo del código, así como:

```
var svg = d3.select("body")
        .append("svg")
        .attr("width", w)   // <-- Acá
```

```
        .attr("height", h); // <-- y aca!
```

Figuras a Partir de Datos

Ya es hora de añadir algunas figuras. Miremos nuestro conjunto de datos de antaño.

```
var dataset = [ 5, 10, 15, 20, 25 ];
```

ahora, se usa `data()` para pasar por cada uno de los datos y así crear un `círculo` por cada uno:

```
svg.selectAll("circle")
   .data(dataset)
   .enter()
   .append("circle");
```

Cabe recordar que `selectAll()` devuelve referencias vacías a todos los `circles` (que aún no existen), `data()` asocia los datos a los elementos que están a punto de crearse, `enter()` devuelve la referencia al espacio que se utiliza para el nuevo elemento y `append()` por último añade un `circle` al DOM.

Para apuntar a todos los `circle` en el futuro, se puede crear una variable nueva que

almacene la referencia a todos:

```
var circles = svg.selectAll("circle")
            .data(dataset)
            .enter()
            .append("circle");
```

Todo bien, pero todos estos círculos necesitan posiciones y tamaños. Prevéngase: El código que va a ver a continuación lo va a descrestar.

```
circles.attr("cx", function(d, i) {
        return (i * 50) + 25;
    })
   .attr("cy", h/2)
   .attr("r", function(d) {
        return d;
   });
```

Deléitese con el ejemplo. Revisemos paso a paso el código.

```
circles.attr("cx", function(d, i) {
        return (i * 50) + 25;
    })
```

Esta parte obtiene la referencia a todos los elementos `circle` y les asigna a cada uno el atributo `cx`. Los datos ya se han asociado a los elementos de tipo `circle`, entonces para cada `circle`, el valor de `d` equivale al valor correspondiente en el conjunto de datos original (5, 10, 15, 20 o 25). Otro valor, `i`, se actualiza automáticamente. `i` es el valor del índice numérico del elemento actual. El contador comienza en 0, por consiguiente para el "primer" círculo `i == 0`, para el segundo círculo `i == 1` y así sucesivamente. La `i` se va a usar para mover cada círculo hacia la derecha, puesto que después de cada paso del loop incrementa el valor de `i`.

```
(0 * 50) + 25 devuelve 25
(1 * 50) + 25 devuelve 75
(2 * 50) + 25 devuelve 125
(3 * 50) + 25 devuelve 175
(4 * 50) + 25 devuelve 225
```

Para garantizar que `i` esté disponible para la función, debe incluirse como un argumento dentro de su definición `(function(d, i))`. También es necesario incluir a `d`, así no se use `d`, dentro de la función (tal como en el caso de arriba).

Ahora, la siguiente línea.

```
'.attr("cy", h/2)
```

`h` es la altura de todo el SVG, entonces `h/2` es la mitad de la altura. Esto tiene el efecto de centrar a todos los círculos desde el punto de vista de alineación vertical.

```
.attr("r", function(d) {
return d;
});
```

Por último, el radio `r` de cada círculo es simplemente asignado a `d` o sea al valor del dato correspondiente.

Uuy, Bonitos Colores

Llenado y trazado de colores son otros de los atributos que se pueden asignar usando los

mismos métodos. Simplemente con añadir este código:

```
.attr("fill", "yellow")
.attr("stroke", "orange")
.attr("stroke-width", function(d) {
    return d/2;
});
```

se obtiene la siguiente página de ejemplo.

Por supuesto que se pueden mezclar atributos y funciones específicas para poder aplicar cualquier combinación de propiedades. El truco con visualizaciones de datos, por supuesto, consiste en escoger *mapeos* apropiados, de tal forma que la expresión visual de sus datos tenga sentido y le sea útil a quién la está viendo.

Siguiente: Tipos de datos –>

Este tutorial fue traducido por Gabriel Coch